Data Structures & Algorithms (DSA)

Intro

Pre-Requisites

Basics of C++

Variables, functions, pointers, ….

OOP using C++

Class, objects, template class, friend function, …

Asympotatic notations

You need to acquaint yourself with Big-O notation, as it plays a pivotal role in evaluating the efficiency of algorithms and data structures.

Dr. Mahmoud Alnamoly

Degrees

- **15** Mid
- **15** Practical (Projects + Practical)
- 10 Oral

COURSE TIMELINE

WEEK 1

Introduction to Data structure and algorithm

WEEK 2

Static Array

WEEK 3

Dynamic Array

WEEK 4

Linked List

Dr. Mahmoud Alnamoly

COURSE TIMELINE

WEEK 5

Stack

Queue

WEEK 6

WEEK 7

Tree

Binary Search Tree

WEEK 8

Dr. Mahmoud Alnamoly

COURSE TIMELINE

WEEK 9

Hash Table

Heap

WEEK 10

WEEK 11

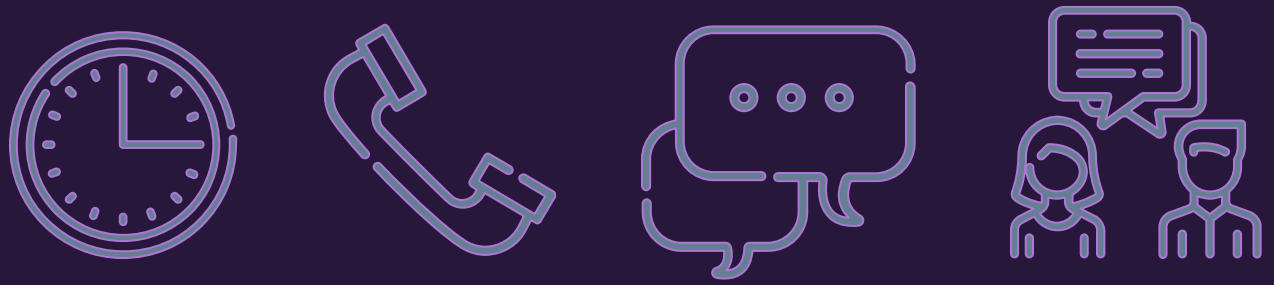
Graph

Strings

WEEK 12

Dr. Mahmoud Alnamoly

Rules



Dr. Mahmoud Alnamoly

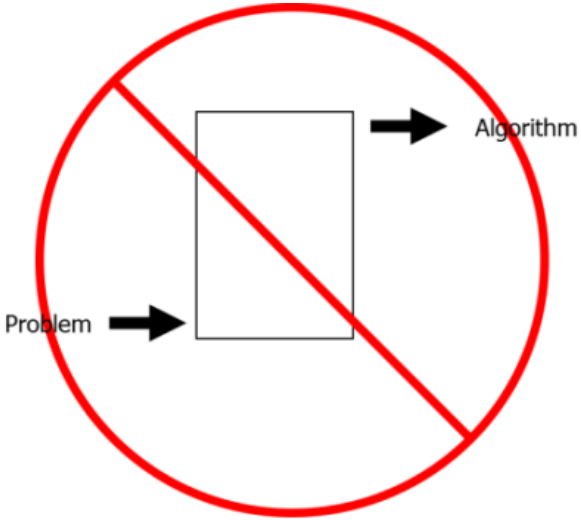
Objective

Understand the fundamental concepts, types and importance of data structures in programming and algorithms analysis.

Dr. Mahmoud Alnamoly

Before we proceed ...

- Problem
- Algorithm
- Program

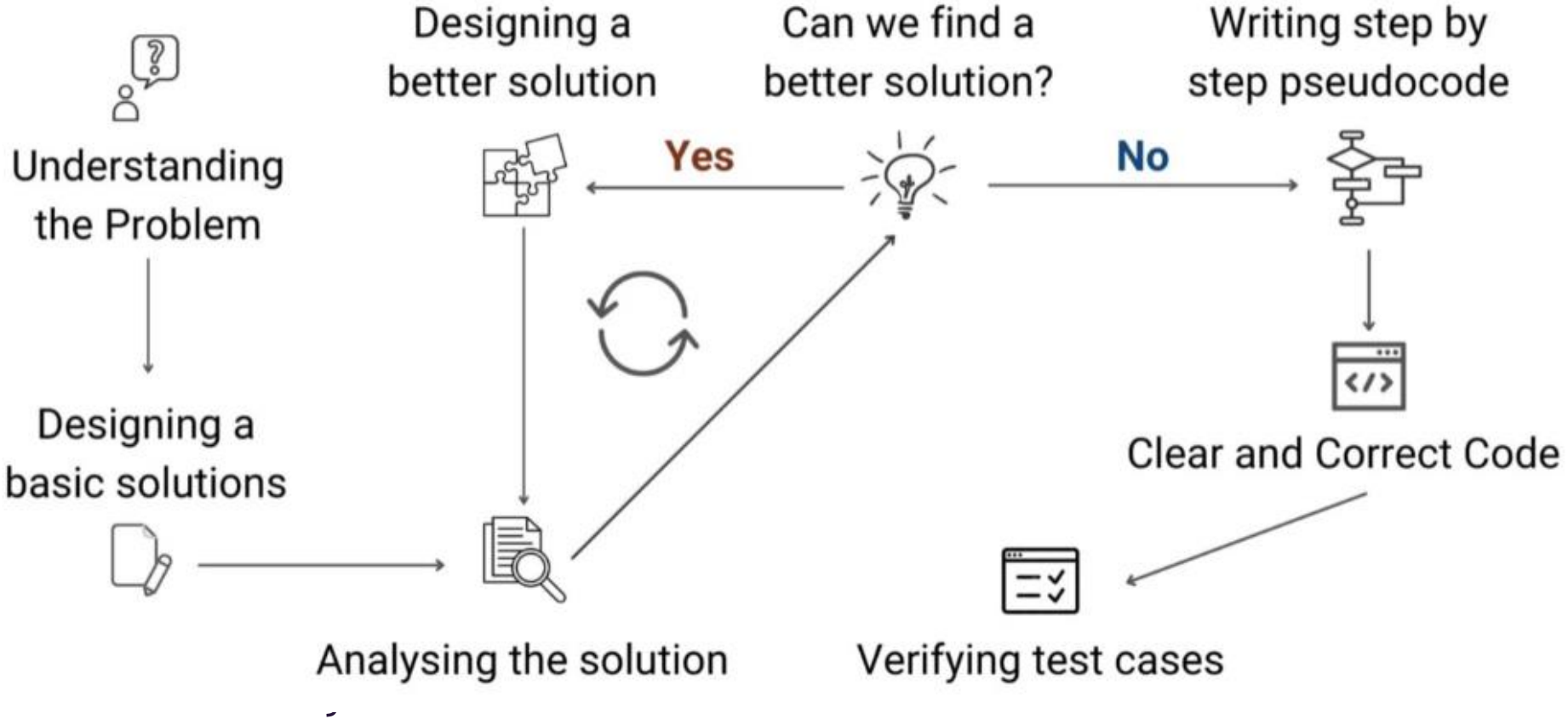


Problem

Algorithm

Input → “Computer” → Output

Problem -> program

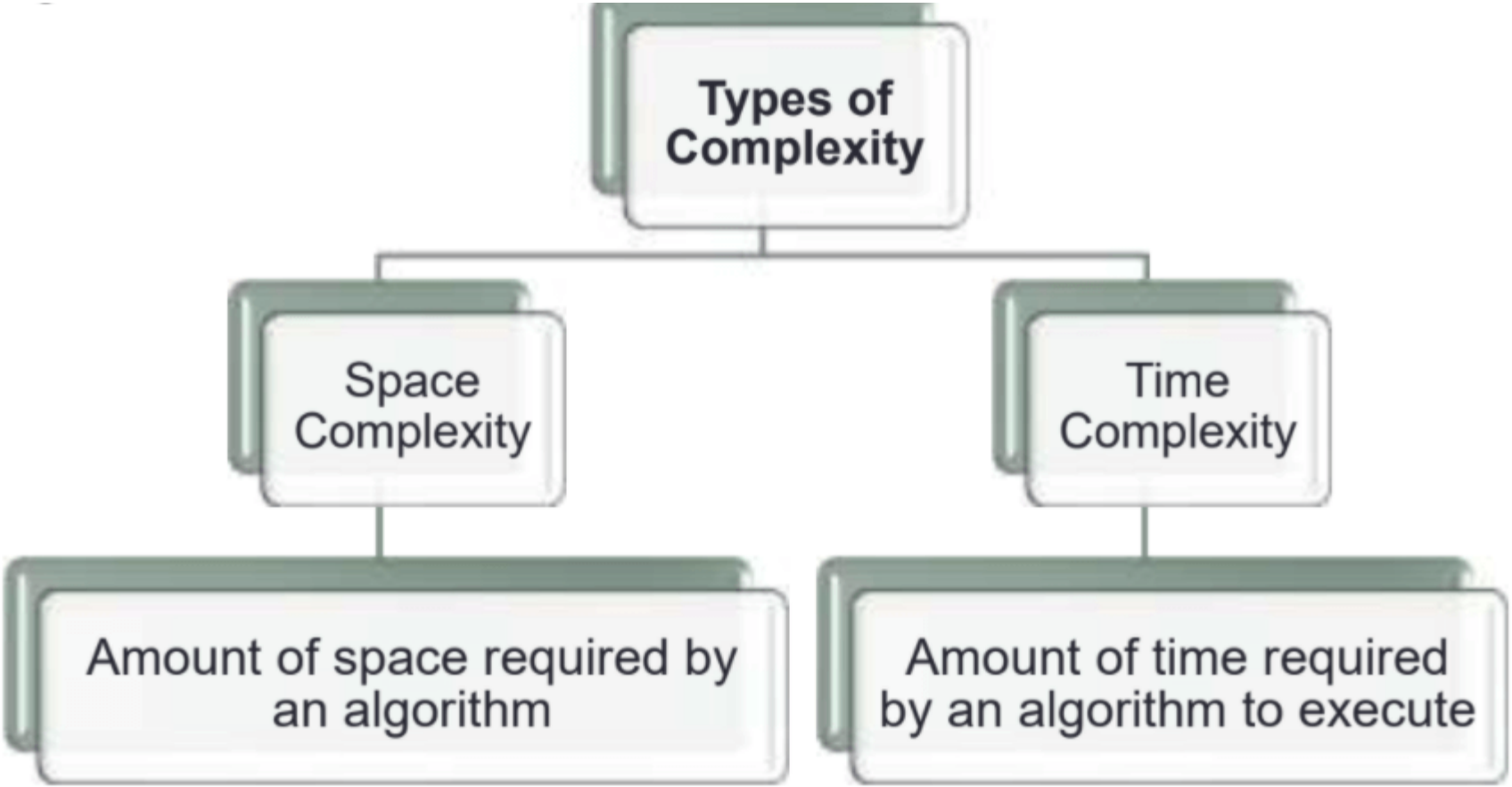


01

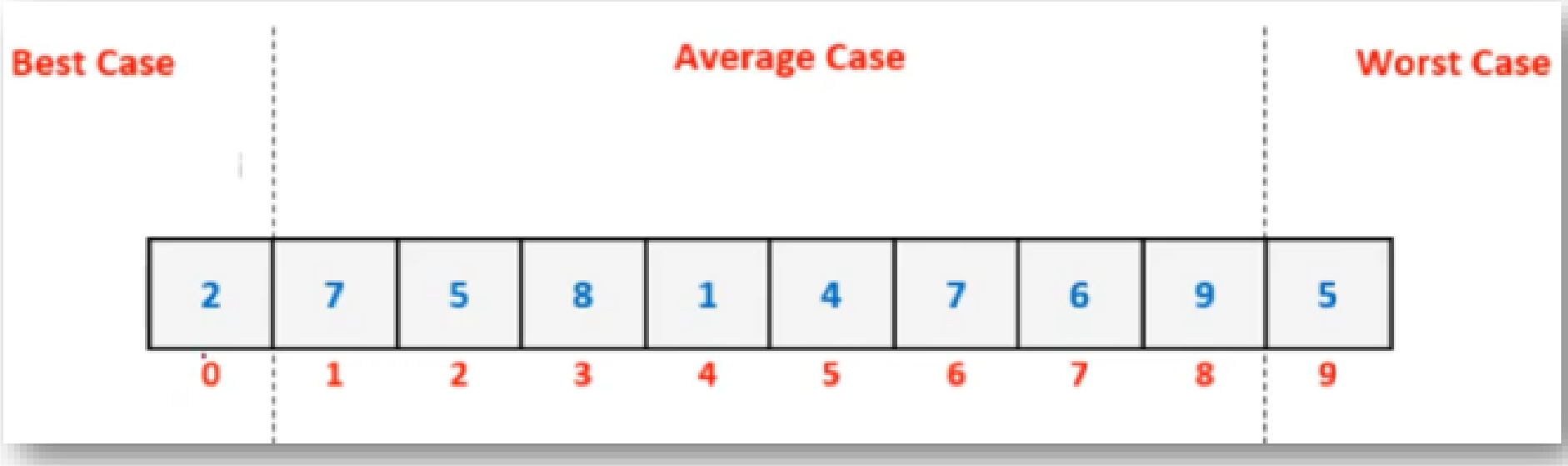
Complexity Analysis

Asympotatic Notaion

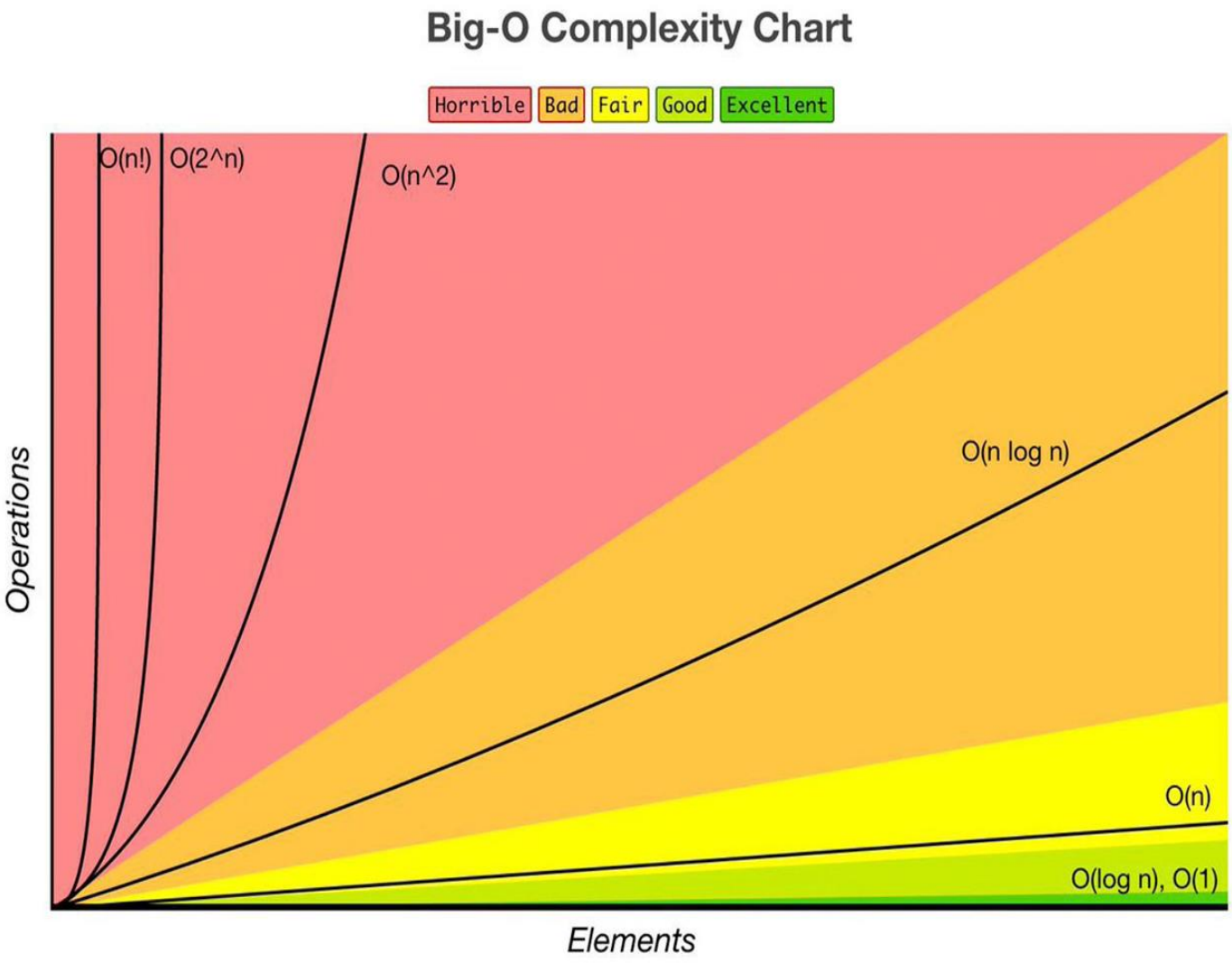
Dr. Mahmoud Alnamoly


Types of Complexity
Space Complexity
Time Complexity
Amount of space required by an algorithm
Amount of time required by an algorithm to execute

Search for a specific item



Ω	Θ	O
Omega	Theta	Big Oh
Lower Bound	Upper & Lower Bound	Upper Bound



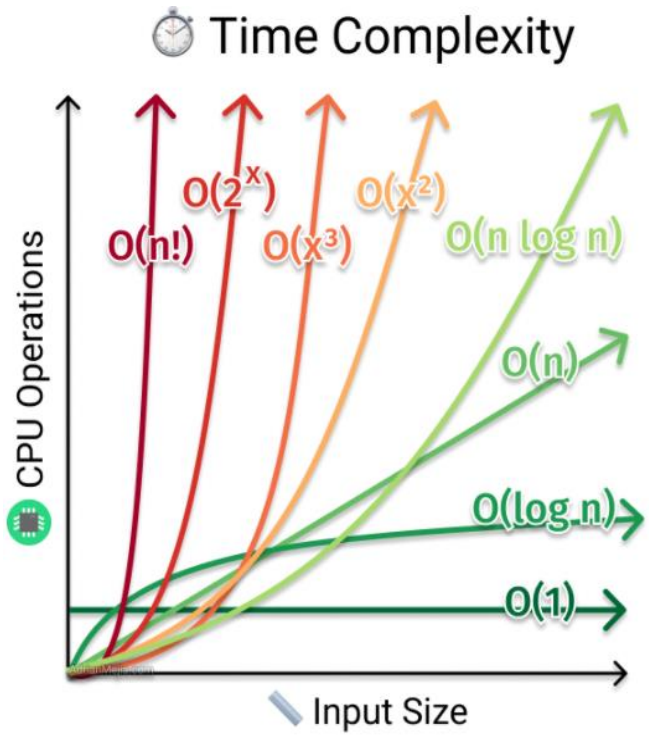
$O(n!), O(c^n), O(n^c)$ - Worst

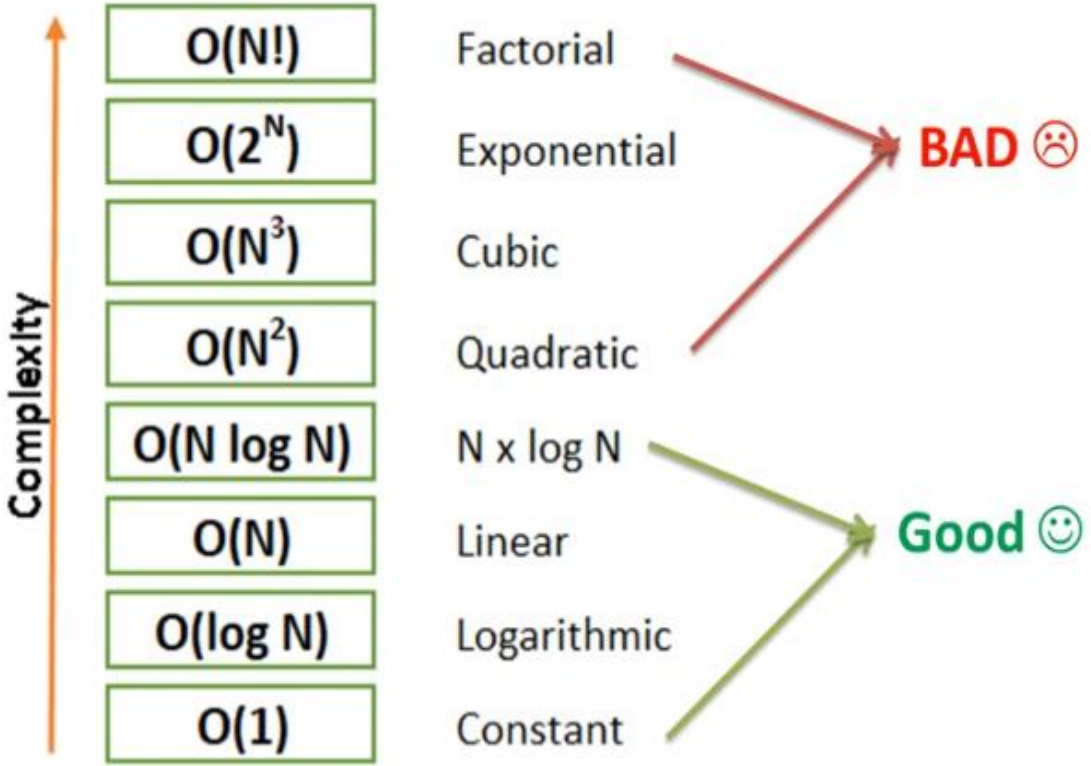
O(nlogn) - Bad

O(n) - Fair

O(logn) - Good

O(1) - Best





Dr. Mahmoud Alnamoly

How to calculate time complexity T(N) of your algorithm ?

Dr. Mahmoud Alnamoly

Method 1:-

Frequency count method

```
Max = A[1]
For I = 1 to N
      If (A[I] > Max)
            Max = A[I]
Return Max
```

Running Time:

Statement	# Operations	Total
`Max = A[1]`	Takes 2 steps: one for = and one for [ ]	2
`For I = 1 to N`	**Initialization :** 1 step **Increment:** N steps (till breaking loop) **Comparison:** N + 1 steps (1 init. & N incr.'s)	2N+2
`If (A[I] > Max)`	Takes 2 steps: one for > and one for [ ]	2N
`Max = A[I]`	Takes 2 steps: one for = and one for [ ]	2N (worst case)
`Return Max`	Takes 1 step	1
	Running Time T(N) =	**6N + 5**

Before going on, there are some **analysis Guiding Principles** that u should war of them:-

- ✓Worst case analysis: over running time bound holds for **every** input of length N
- ✓focus on running time for **large** input size N
- ✓Don't pay much attention to **constant factors**

Running time "Exact Time" Vs. Order "Asymptomatic time"

Constants are not important…(**Ex.** $10^6 \log_2 N$ vs. $20 N$ *compare N=8, 10^6, 10^9*)

Biggest order is the dominant (**Ex.** $N^2 + N \log N + 10 N$ *compare N=10, 1000*)

Method 2:-

Asymptomatic Time

There are some rules to calculate the order "Asymptotic time"

- ✓Statements: order of 1
- ✓Conditions: choose the max
- ✓Loops: order of body × # of iterations

```
Sum = 0
For I = 1 to 1000
        Sum = Sum + I
End for
```

$T(N) = \mathbf{O}(1)$

```
Max = A[1]
For I = 1 to N
        If (A[I] > Max)
                Max = A[I]
Return Max
```

$T(N) = \mathbf{O}(n)$

```
Switch (choice)
Case 1:                    O(1)
        Print N
Case 2:                    O(N)
        MaxItem = GetMax(A, N)
        Print MaxItem
Case 3:                    O(N²)
        Sum = 0
        For I = 1 to N
                If (Sum < GetMax(A, N))
                        Sum = Sum + A[I]
                End if
        End for
        Print Sum
End Switch
```

$$T(N) = \mathbf{O}(n^2)$$

Linear Search

```
Found = false
I = 0
While (I < N and Not Found)
        If Item == A[I] then
                Found = true
        End If
        I = I + 1
End While
Return Found
```

$$T(N) = \mathbf{O}(n)$$

Method 2:-

Asymptomatic Time

Using Equations

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \approx \frac{N^2}{2}$$

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} \approx \frac{N^3}{3}$$

$$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$$

$$\sum_{i=0}^{N} k^i = \frac{k^{N+1} - 1}{k-1}$$

$$\sum_{i=1}^{lgn} 2^i = O(2n-1)$$

$$\sum_{i=1}^{lgn} any\ f(n) = O(lgn * f(n))$$

$$\sum_{i=1}^{n} any\ f(n) = O(n * f(n))$$

Example 7

```
int x = 1 + 3;
for(int i = 1; i ≤ n; i + +)
    print i;
int y = x * 120;
print y;
```

$T(N) = \mathbf{O}(n)$

Example 8

```
for(int i = n; i ≥ 1; i − −)
    print i;
```

$T(N) = \mathbf{O}(n)$

Example 9

```
for(int i = 1; i ≤ n; i *= 2)
    print i;
```

$T(N) = \mathbf{O}(log_2 n)$

Example 10

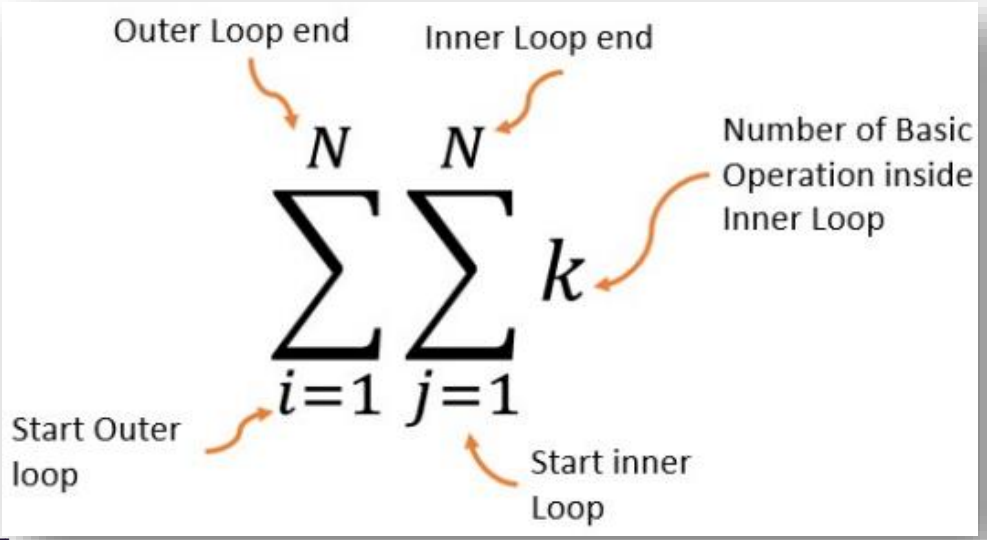
```
for(int i = n; i ≥ 1; i /= 2)
    print i;
```

$T(N) = \mathbf{O}(log_2 n)$

Example 11

```
for(int i = 1; i ≤ n; i++)
    for(int j = 1; j ≤ n; j++)
        print "hello";
```

$$T(N) = \mathbf{O}(n^2)$$



Example 12

```
for(int i = 1; i ≤ n; i++)
    for(int j = 1; j ≤ n; j++)
        print "hello";

for(int i = 1; i ≤ n; i *= 3)
    print "hello";
```

$$T(N) = \mathbf{O}(n^2 + log_3 n)$$

Example 13

```
for(int i = 1; i ≤ n; i *= 3)
    for(int j = 1; j ≤ n; j++)
        print "hello";
```

$$T(N) = \mathbf{O}(n\ log_3 n)$$

Example 14

```
for(int i = 1; i ≤ n; i *= 3)
    for(int j = 1; j ≤ n; j *= 3)
        print "hello";
```

$$T(N) = \mathbf{O}(log_3{}^2\, n)$$

Example 15

```
for(int i = 1; i ≤ n; i++)
    for(int j = 1; j ≤ i; j++)
        print "hello";
```

$$T(N) = \mathbf{O}(??)$$

Data Structure ?

Dr. Mahmoud Alnamoly

Data Structure

A data structure is a scheme for organizing data in the memory of a computer for efficient storage and efficient manipulation.

The idea is to reduce the space and time complexities of different tasks.

Dr. Mahmoud Alnamoly

Algorithm, Program, Data Structure

Before a program is written, we should have a good idea of how to accomplish the task being implemented by this program. But, we should be more concerned with what the program should do, not how it should or could be done.

Behavior of the program is more important than the mechanism accomplishing it.

We should know what are the operations of my program? These operations for example, modifying, searching for some details, or storing something.
After these operations are specified, the implementation of the program may start.

The implementation decides which data structure should be used to make execution most efficient in terms of **time** and **space**.

The best data structure for a problem is the one that allows the computer program to solve the problem efficiently in terms of the required storage and time.

WHY ?

Why study data structure ?

Understanding data structures will boost your problem-solving skills and enhance your programming thinking ability. **Let's break this down:**

Efficiency

Efficient software or applications are every programmer's goal. By using the appropriate data structure, you can drastically improve the efficiency of your code.

Problem-solving Skills

Once you master data structures, you'll find it much easier to solve complex problems. It'll be like having a well-organized toolbox, where you know exactly what tool to use and when.

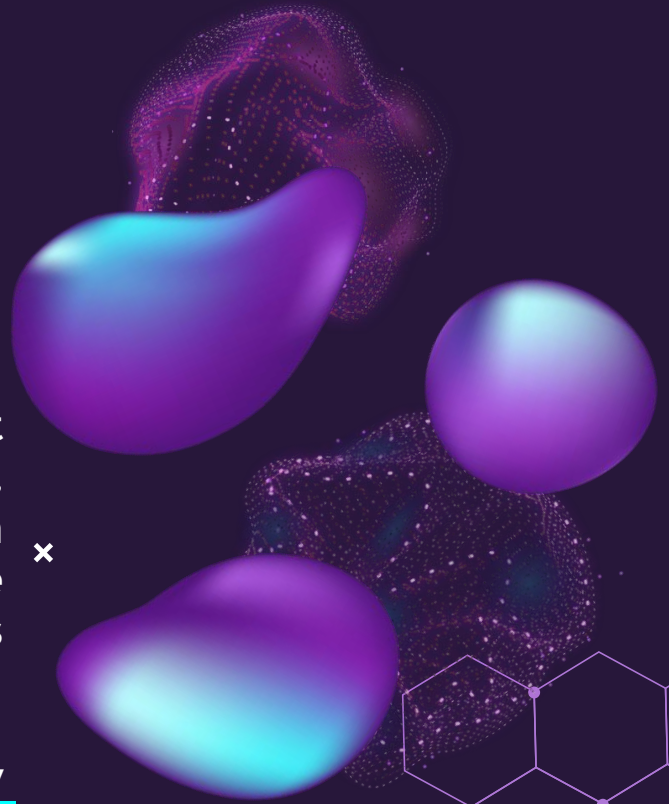
Career Advancement

Regardless of the programming language you work with, data structures are fundamental to computer science. A strong understanding of data structures can significantly improve your career prospects.

Let's visualize this concept further.

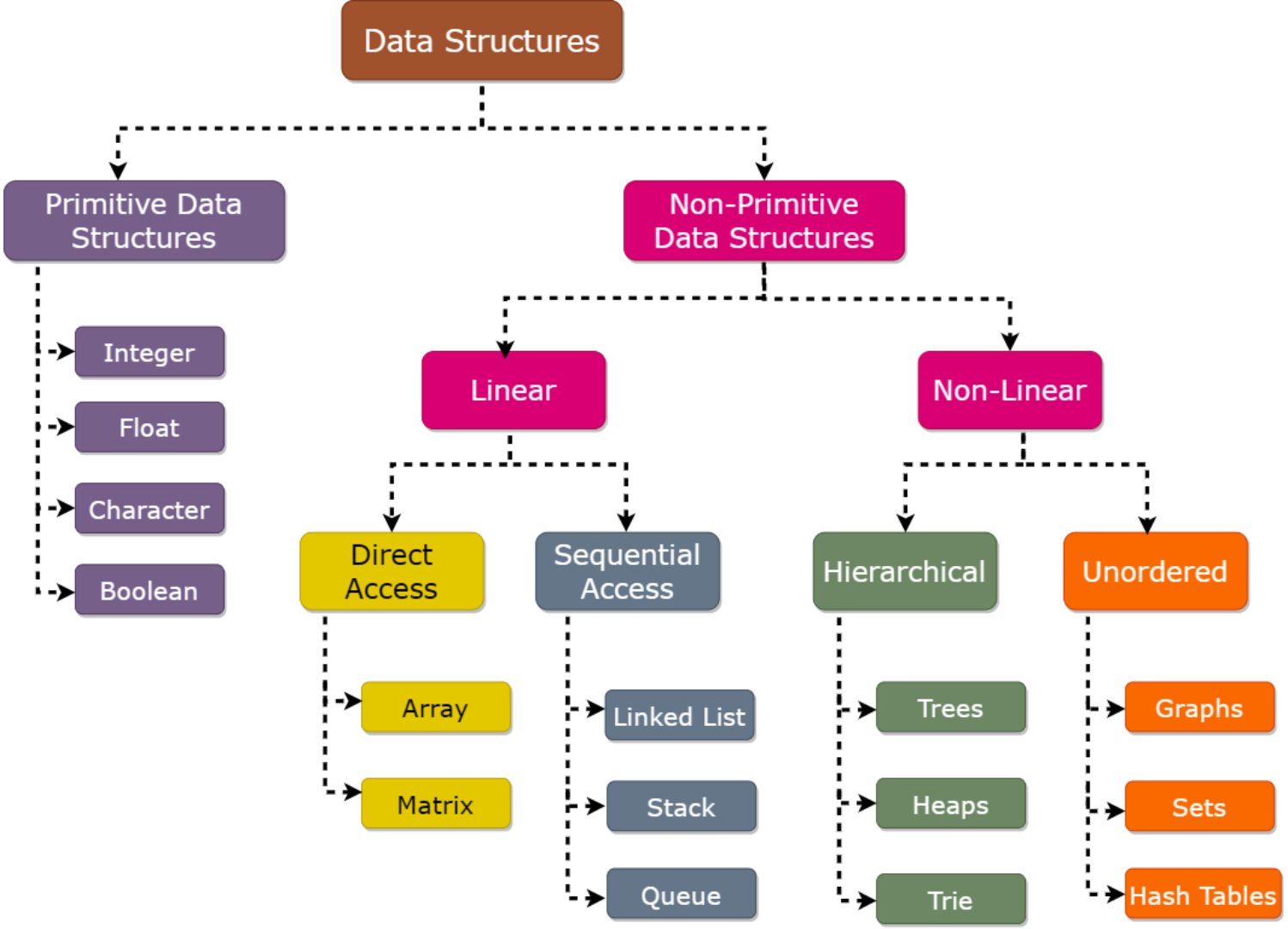
Consider a library filled with books. Without any particular structure or arrangement, finding a book in this library would be a challenging task. But, if we arrange these books by genre, author, or title, it becomes much easier to locate a specific book.

This is the essence of data structures - they help us manage, organize, and use data more effectively.



DS TYPES ?

Dr. Mahmoud Alnamoly


Data Structures
Primitive Data Structures
Non-Primitive Data Structures
Integer
Float
Character
Boolean
Linear
Non-Linear
Direct Access
Sequential Access
Hierarchical
Unordered
Array
Matrix
Linked List
Stack
Queue
Trees
Heaps
Trie
Graphs
Sets
Hash Tables

Data Structures Types

Basically, data structures are divided into :

Linear data structure

The data items are arranged in sequential order, one after the other, Implementation is easy.

Non-linear data structure

The data items are arranged in non-sequential order (hierarchical manner), Implementation is difficulty.

Data Structures Types

Basically, data structures are divided into :

Static data structure

- In Static data structure the size of the structure is fixed.
- The content of the data structure can be modified but without changing the memory space allocated to it.

Dynamic data structure

- In Dynamic data structure the size of the structure in not fixed and can be modified during the operations performed on it.
- Dynamic data structures are designed to facilitate change of data structures in the run time.

Data Structures Types

Basically, data structures are divided into :

Homogeneous data structure

All the elements in Homogeneous data structure are the same such as array

Non-Homogeneous data structure

The elements in Non-Homogeneous data structure may or may not be the same such as structures

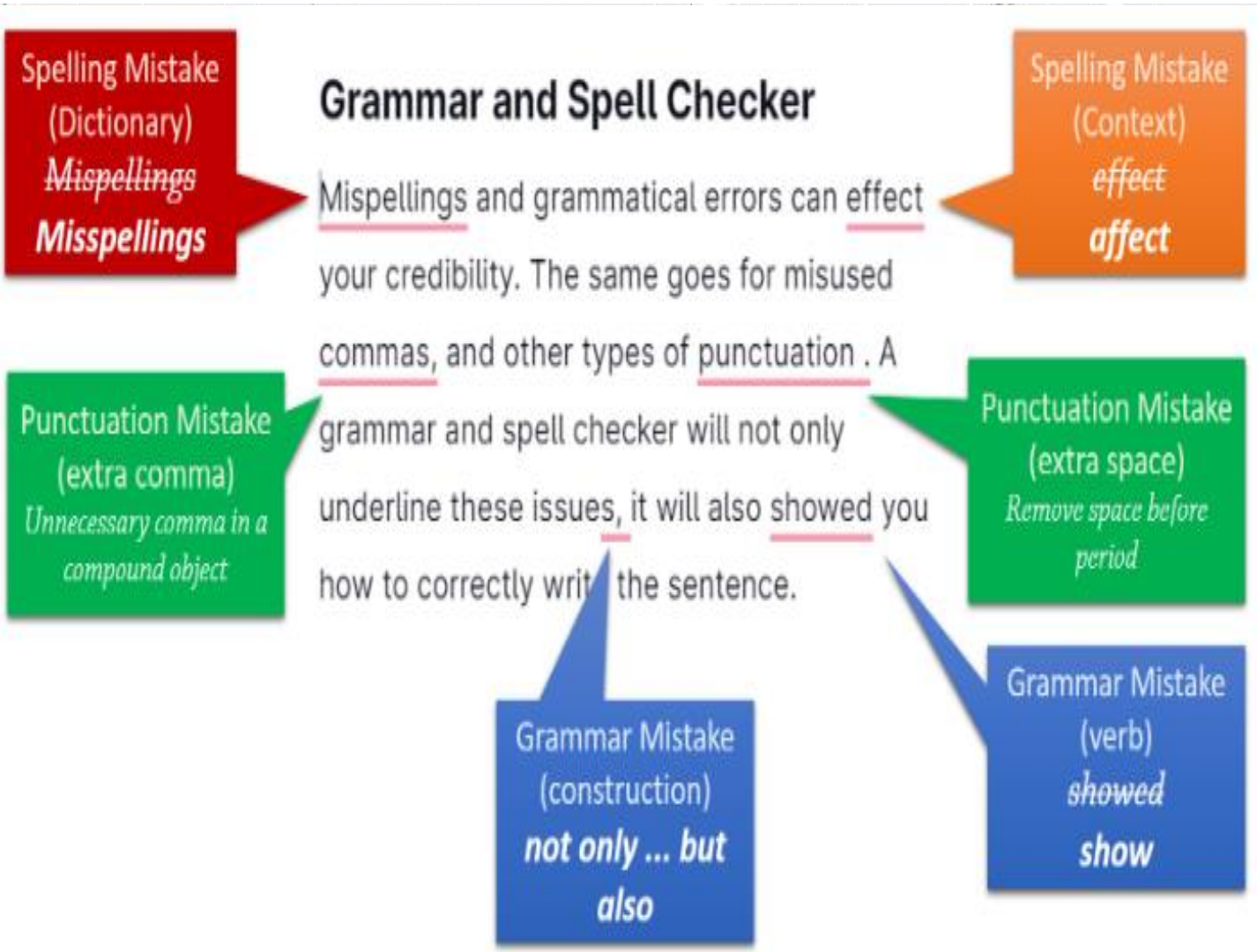
Motivations

Dr. Mahmoud Alnamoly

Motivations

Data Structure:

1. Array
2. Stack
3. Queue
4. Linked list
5. Tree
6. Graph
7. Hashing


Grammar and Spell Checker
Mispellings and grammatical errors can effect your credibility. The same goes for misused commas, and other types of punctuation . A grammar and spell checker will not only underline these issues, it will also showed you how to correctly writ the sentence.
Spelling Mistake (Dictionary)
Mispellings
Misspellings
Spelling Mistake (Context)
effect
affect
Punctuation Mistake (extra comma)
Unnecessary comma in a compound object
Punctuation Mistake (extra space)
Remove space before period
Grammar Mistake (construction)
not only ... but also
Grammar Mistake (verb)
showed
show

Notes

- Choosing the right data structure depends on the type of data you're working with and the operations you need to perform.
- The choice of a data structure can significantly impact the performance, scalability, and maintainability of a software system.
- We can build any data structure with a specific characteristic and operations.
- **ADT** comes first, and the implementation of the required data structure comes next.

ADT

Dr. Mahmoud Alnamoly

Abstract Data Type

Abstract data type defines a data type in terms of a type and a set of operations on that type.

An abstract data type (ADT) is the way we look at a data structure, focusing on what it does and ignoring how it does its job.

Note:-

ADT definition does not specify how the data type is implemented.

Data structure is the physical implementation of an ADT.

Stack ADT

A Stack contains elements of same type arranged in sequential order. All operations takes place at a single end that is top of the stack and following operations can be performed:

push() – Insert an element at one end of the stack called top.

pop() – Remove and return the element at the top of the stack, if it is not empty.

peek() – Return the element at the top of the stack without removing it, if the stack is not empty.

size() – Return the number of elements in the stack.

isEmpty() – Return true if the stack is empty, otherwise return false.

isFull() – Return true if the stack is full, otherwise return false.

Choosing the Right Data Structure:

Selecting the appropriate data structure requires a deep understanding of your data and the operations you intend to perform. Here's a step-by-step guide:

1. Analyze Your Data:

Start by understanding the nature of the data you'll be working with. Is it structured or unstructured? Does it have unique characteristics like order, duplications, or relationships?

2. Identify Operations:

Determine the primary operations your application needs to perform on the data, such as searching, insertion, deletion, and traversal.

Choosing the Right Data Structure:

3.Time Complexity:

Evaluate the time complexity of the chosen data structure for your operations. Opt for structures that offer the most efficient algorithms for your use case.

4. Space Complexity:

Assess the memory requirements of your data structure. Ensure it's suitable for the available hardware and scalability needs.

5. Scalability:

Think about the potential growth of your data. Choose a data structure that can handle increasing data volumes without significant performance degradation.

Choosing the Right Data Structure:

6. Built-In Libraries:

In many programming languages, there are built-in libraries that provide optimized data structures for common use cases. Utilize these libraries to save development time and ensure efficiency.

7. Testing and Benchmarking:

Finally, thoroughly test your chosen data structure under realistic conditions to ensure it meets your performance expectations. Benchmarking can help identify bottlenecks and fine-tune your choice.

Static Array

>>> Next Lecture